NUSC Technical Memorandum 851062
22 May 1985

# Mathematically Equivalent, Computationally Non-equivalent Formulas and Software Comprehensibility

Marvin J. Goldstein
Surface Ship Sonar Department




Naval Underwater Systems Center
Newport, Rhode Island/New London, Connecticut

# Report Documentation Page

*Form Approved*
*OMB No. 0704-0188*



| 1. REPORT DATE | 2. REPORT TYPE | 3. DATES COVERED |
|---|---|---|
| **22 MAY 1985** | **Technical Memo** | **22-05-1985 to 22-05-1985** |

| 4. TITLE AND SUBTITLE | 5a. CONTRACT NUMBER |
|---|---|
| **Mathematically Equivalent, Computationally Non-equivalent Formulas and Software Comprehensibility** | 5b. GRANT NUMBER |
| | 5c. PROGRAM ELEMENT NUMBER |
| 6. AUTHOR(S) **Marvin Goldstein** | 5d. PROJECT NUMBER **771Y00** |
| | 5e. TASK NUMBER |
| | 5f. WORK UNIT NUMBER |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Underwater Systems Center,New London,CT,06320** | 8. PERFORMING ORGANIZATION REPORT NUMBER **TM 851062** |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSOR/MONITOR'S ACRONYM(S) |
| | 11. SPONSOR/MONITOR'S REPORT NUMBER(S) |

12. DISTRIBUTION/AVAILABILITY STATEMENT
**Approved for public release; distribution unlimited**

13. SUPPLEMENTARY NOTES
**NUWC2015**

14. ABSTRACT

**In the development of mathematical software, often the formula that defines the mathematical purpose of the software is not used directly in the software. The computational algorithm used is often mathematically equivalent to the defining formula, but bears little resemblance to it for computational reasons. Therefore, although the flow of control of the coded algorithm may be visible to a maintenance programmer, comprehending and maintaining the code effectively may still be difficult if only the formula that defines the purpose of the code is provided as documentation. In this memorandum, we provide some examples of this consequence of transforming the mathematical definition of a computation into a coded algorithm.**

15. SUBJECT TERMS
**mathematical software**

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | |
| **unclassified** | **unclassified** | **unclassified** | **Same as Report (SAR)** | **17** | |

TM 851062

NAVAL UNDERWATER SYSTEMS CENTER
NEW LONDON LABORATORY
NEW LONDON, CONNECTICUT 06320

Technical Memorandum

# MATHEMATICALLY EQUIVALENT, COMPUTATIONALLY NON-EQUIVALENT FORMULAS AND SOFTWARE COMPREHENSIBILITY

Date: 22 May 1985

Prepared by: Marvin J. Goldstein
Marvin J. Goldstein
Surface Ship Sonar Dept

## ABSTRACT

In the development of mathematical software, often the formula that defines the mathematical purpose of the software is not used directly in the software. The computational algorithm used is often mathematically equivalent to the defining formula, but bears little resemblence to it for computational reasons. Therefore, although the flow of control of the coded algorithm may be visible to a maintenance programmer, comprehending and maintaining the code effectively may still be difficult if only the formula that defines the purpose of the code is provided as documentation. In this memorandum, we provide some examples of this consequence of transforming the mathematical definition of a computation into a coded algorithm.



## ACKNOWLEDGEMENT

The author is grateful to Dr. David Wood for useful comments in the preparation of this memorandum.



## ADMINISTRATIVE INFORMATION

This memorandum was prepared under Job Order No. 771Y00, Special Projects and Studies, and Job Order No. W65000, EVA Prog. Modeling Efforts, Principal Investigator, H. Weinberg (Code 3332). The author of this memorandum is located at the New London Laboratory, Naval Underwater Systems Center, New London, Ct 06320.

## INTRODUCTION

It is generally agreed that clearly written code helps the programmer working on it to grasp its meaning more quickly, so that program changes can be applied with more confidence. Although structured coding promotes program readability by exposing a program's flow of control, so that one can follow the control flow in a top-down manner, it does not necessarily follow that a reader of structured code will comprehend its algorithms sufficently to effectively maintain the code. The reason for this is that in general there are a multitude of algorithms that can be chosen to perform any particular program transaction. The algorithm that is chosen is determined by computational considerations, and often bears little resemblence to the formulas that were used to define the transaction originally. Therefore, although the coded algorithm's flow of control may be highly visible to a maintenance programmer, comprehending the algorithm may be difficult without documentation that describes the algorithm and its computational advantages, as well as identifying the transaction it represents.

In this memorandum, we give some examples of mathematically - but not computationally - equivalent formulas that result from transforming the mathematical definition of a computation into a coded algorithm, showing that without documenting the algorithm the resulting structured code may not be comprehensible enough to maintain effectively.

## BACKGROUND INFORMATION

In the development of mathematical software, often the formula that defines the mathematical purpose of the software is not used directly in the software. Often the computational algorithm is based on some mathematically equivalent formula that is determined by computer arithmetic, operating system or hardware features that impact computational accuracy, execution efficiency or storage economy. For example, since the laws of additive associativity and closure for the real number system do not hold for floating-point computer number systems, the mathematically equivalent expressions $(x+1)-1$ and $x+(1-1)$ will not give the same computer results for all values of $x$.

Furthermore, the same algorithm can be implemented in a computer program in different ways. For example, the structure of the flow of control of a program module depends on the programmer. In particular, given two FORTRAN implementations of an algorithm, the flow of control of one of them may be easy to follow, while the flow of control of the other may be as difficult to follow as the entwining strands of sphagetti [1]. On the other hand, implementation efforts to improve program excution efficiency, which refine the algorithm further, may demote software clarity.

Therefore, three stages to the development of mathematical software can be identified. Firstly, a formula is specified that defines the purpose of the computation. Secondly, a computational algorithm is selected from among mathematically equivalent forms of the defining formula that differ in their computational performance. This is done to produce an algorithm that satisfies certain computational requirements of accuracy, execution efficiency and/or storage efficiency. Thirdly, the selected algorithm is coded. Therefore, unless the mapping of the defining formula into the coded algorithm is documented, maintenance of the software may still be difficult regardless of the structuredness of the control flow of the code.

## EXAMPLES

Consider constructing software to compute the magnitude of a complex number z = x + iy on a computer where the largest and smallest positive computer numbers are n and 1/n, respectively, with n >> 2. The standard mathematical definition for this computation is

$$abs(z) = sqrt(x^2 + y^2) \qquad (1)$$

If either the magnitude of x or the magnitude of y is outside the interval [1/sqrt(n), sqrt(n)], use of the standard definition as a computational formula results in computer overflow or underflow. On the other hand, if the magnitude of x and the magnitude of y lie in the interval [1/n, n/sqrt(2)], which contains the interval [1/sqrt(n), sqrt(n)],using the mathematically equivalent formula

$$abs(z) = v\ sqrt(1 + (w/v)^2) \qquad (2)$$

where

$$v = max(abs(x), abs(y)), \quad w = min(abs(x), abs(y))$$

does not cause computer overflow and makes computer underflow inconsequential. However, note that in Figure 1 the structured FORTRAN codes based on these mathematically equivalent formulas only vaguely resemble each other. In fact, given that the purpose of Code 2 is to compute the magnitude of a complex number z, which is traditionally defined by Eq. (1), it is likely that without additional information a maintenance programmer would have difficulty comprehending the encoded algorithm by just reading the FORTRAN code.

FIGURE 1

CODE 1 for Eq.(1)

```
ABSZ = -1.0
ROOT = SQRT(N/2.0)
B = ABS(X).LT.ROOT .AND. ABS(Y).LT.ROOT
IF(B)THEN
   ABSZ = SQRT(X**2 + Y**2)
END IF
```

CODE 2 for Eq.(2)

```
 ABSZ = -1.0
 ROOT = N/SQRT(2.0)
 W = AMIN1(ABS(X),ABS(Y))
 V = AMAX1(ABS(X),ABS(Y))
 IF(V.EQ.0.0)THEN
    ABSZ = 0.0
 ELSE
    IF(V.LT.ROOT)
+      ABSZ = V*SQRT(1.+(W/V)**2)
 END IF
```

As a second example, consider computing the matrix product V of the n-th order matrices A and X, which is defined by

$$(V_{ik}) = \left(\sum_{j=1}^{n} A_{ij}\ X_{jk}\right) \qquad (3)$$

The mathematical definition of the element in the i-th row and k-th column of V, given by (3), is the inner product of the i-th row of matrix A with the k-th column of matrix X. Typically, one computes all the elements in the k-th column in this way for each of the columns of the product V. Expressing this in structured FORTRAN gives

```
          CODE 3

      DO 30 K = 1, N
         DO 25 I = 1,N
            V(I,K) = 0.0
            DO 15 J = 1, N
               V(I,K) = V(I,K) + A(I,J) * X(J,K)
15          CONTINUE
25       CONTINUE
30    CONTINUE
```

Note that for each value of k this algorithm visits the elements of matrix A in row major order:

$$A_{11}, A_{12}, \ldots, A_{1n}, A_{21}, A_{22}, \ldots, A_{2n}, \ldots, A_{n1}, A_{n2}, \ldots, A_{nn}$$

But, in [2], it is shown that on virtual memory systems like the VAX, addressing matrix elements in this order is inefficient when assigned main memory is too small to contain the code and its matrices. In order to reduce execution time, matrix algorithms written in FORTRAN should be based on addressing matrix elements in column major order:

$$A_{11}, A_{21}, \ldots, A_{n1}, A_{12}, A_{22}, \ldots, A_{n2}, \ldots, A_{1n}, A_{2n}, \ldots, A_{nn}$$

Now consider the following mathematically equivalent method of computing the elements in the k-th column of matrix V, which is based on visiting the elements of the A array in column major order. The k-th column of V is computed recursively by generating a finite sequence of n + 1 vector approximations to it:

$$V_k^{(0)} = 0, \quad V_k^{(j)} = V_k^{(j-1)} + X_{jk} A_j \qquad (j = 1, \ldots, n) \qquad (4)$$

where

$$A_j = \begin{bmatrix} A_{1j} \\ A_{2j} \\ \vdots \\ A_{nj} \end{bmatrix}$$

$(n)$
and $V_k$ is the k-th column of V; hence at the j-th stage of the recursion each element of the j-th column of matrix A is multiplied by the j-th element of column k of matrix X, so that the algorithm visits the elements of matrix A in column major order. The corresponding structured FORTRAN code for this is

CODE 4

```
      DO 10 K = 1, N
         DO 5 I = 1, N
            V(I,K) = 0.0
 5       CONTINUE
10    CONTINUE
      DO 30 K = 1, N
        DO 25 J = 1, N
           DO 15 I = 1, N
              V(I,K) = V(I,K) + A(I,J) * X(J,K)
15         CONTINUE
25      CONTINUE
30    CONTINUE
```

In Table 1, VAX 11/780 batch execution times for Codes 3 and 4 show the superiority in execution efficiency of Code 4 when matrix memory requirements exceed a users assigned main memory extent of 250 pages. It is assumed that V(I,K) is accumulated in double precision.

TABLE 1

TIMING FOR MATRIX MULTIPLICATION ON THE VAX 11/780
(assigned main memory extent = 32,000 words)

| Order (n) | Memory Requirements (in words) | Code 3 ( in cpu sec. ) | Code 4 ( in cpu sec. ) |
|---|---|---|---|
| 16 | 1,024 | .07 | .07 |
| 32 | 4,096 | .61 | .53 |
| 64 | 16,384 | 5.75 | 4.32 |
| 128 | 65,536 | 86.41 | 45.91 |
| 200 | 160,000 | 910.88 | 157.78 |
| 240 | 230,400 | 5581.4 | 274.72 |
| 256 | 262,144 | 6828.3 | 362.44 |
| 512 | 1,048,576 | 56858.0 | 2668.27 |

Unfortunately, knowing only the usual definition (3) of matrix multiplication and that Code 4 computes the product of two n-th order matrices, a maintenance programmer might replace Code 4 with the shorter Code 3 in order to reduce program control complexity. However, doing this could diminish the code's execution efficiency dramatically. In other words, failure to provide documentation that describes the computational ramifications of the algorithm used to implement a program module's function can lead to counter-productive code modifications during software maintenance.

Now consider unrolling the innermost DO-loops of Code 4 to reduce the loop overhead of incrementing the value of I, testing the new value of I against N and branching to the beginning of the loop. Doing this will improve the program's execution efficiency and refine algorithm (4) further. Assuming that 4 divides N exactly, unrolling Code 4 to a depth of 4 gives the more efficient, but longer structured code

CODE 5

```
      DO 10 K = 1, N
         DO 5 I = 1, N, 4
            V(I,K) = 0.0
            V(I+1,K) = 0.0
            V(I+2,K) = 0.0
            V(I+3,K) = 0.0
 5       CONTINUE
10    CONTINUE
      DO 30 K = 1, N
         DO 25 J = 1, N
            DO 15 I = 1, N, 4
               V(I,K) = V(I,K) + A(I,J) * X(J,K)
               V(I+1,K) = V(I+1,K) + A(I+1,J) * X(J,K)
               V(I+2,K) = V(I+2,K) + A(I+2,J) * X(J,K)
               V(I+3,K) = V(I+3,K) + A(I+3,J) * X(J,K)
15          CONTINUE
25       CONTINUE
30    CONTINUE
```

The corresponding algorithm for computing the k-th column of V is

$$V_k^{(j,i)} = V_k^{(j-1,i)} + A_j^{(i)} X_{jk} \qquad (j = 1, \ldots, n) \qquad (5)$$

where for each value of j

$$V_k^{(j,i)} = \begin{bmatrix} V_{4i-3,k}^{(j)} \\ \vdots \\ V_{4i,k}^{(j)} \end{bmatrix} \qquad A_j^{(i)} = \begin{bmatrix} A_{4i-3,j} \\ \vdots \\ A_{4i,j} \end{bmatrix} \qquad (i=1,\ldots,n/4)$$

Note that without an explanation of loop unrolling, a maintenance programmer may puzzle over the structured FORTRAN implementation of algorithm (4) given by Code 5.

As a third example, consider computing the eigenvalues of a real symmetric Toeplitz matrix. The elements of a symmetric Toeplitz matrix T of order n satisfy the relationship

$$T_{jk} = T_{j+1,k+1} = T_{kj} \qquad (6)$$

Therefore,

$$T_{jk} = T_{n-k+1,n-j+1} = T_{n-j+1,n-k+1} \qquad (7)$$

by adding n+1-j-k to both j and k. Hence reversing the order of the columns and rows of a symmetric Toeplitz matrix yields the original matrix; in matrix notation

$$JTJ = T \qquad (8)$$

where J is obtained by reversing the columns of the n-th order identity matrix I. A consequence of (8) is that any real symmetric Toeplitz matrix T of even order 2n can be written in the form

$$T = \begin{bmatrix} A & BJ \\ JB & JAJ \end{bmatrix} \qquad (9)$$

where $A^t = A$, $B^t = B$, A is Toeplitz and B is Hankel. But note that matrix T is similar to

$$R^t T R = \begin{bmatrix} A+B & 0 \\ 0 & A-B \end{bmatrix} \qquad (10)$$

where R is the orthogonal matrix

$$R = \text{sqrt}(.5)\begin{bmatrix} I & I \\ J & -J \end{bmatrix} \qquad (11)$$

Therefore, the eigenvalues of matrices T and $R^t T R$ are the same. Consequently, computing the eigenvalues of an even order real symmetric Toeplitz matrix T is mathematically equivalent to computing the eigenvalues of the smaller symmetric matrices A+B and A-B, where the elements in the k-th column of matrix A+B are given by

$$(A+B)_{jk} = T_{jk} + T_{j,2n-k+1}$$

$$= T_{1,k-j+1} + T_{1,2n+2-j-k} \quad \text{if } j \le k \qquad (12)$$

$$= T_{j-k+1,1} + T_{2n+2-j-k,1} \quad \text{if } j > k$$

for k=1,...,n.

Now consider constructing software that computes the eigenvalues of a real symmetric Toeplitz matrix T of even order n by using a canned subroutine that computes the eigenvalues of any real symmetric matrix, where by definition (6)

$$T_{jk} = T_{1,k-j+1} \quad \text{if } j \leq k \qquad (13)$$

$$= T_{1,j-k+1} \quad \text{if } j > k$$

Given the first row TR1 of matrix T, two possible structured FORTRAN subroutines that compute matrix T's eigenvalues EIG using a library subroutine EIGRS for the eigenvalues of a real symmetric matrix are

CODE 6

```
      SUBROUTINE TOPEIG(TR1,N,T,EIG)
      DIMENSION T(N,N),EIG(N),TR1(N)
      DO 20 K = 1,N
         DO 10 J = 1,N
            IF(J.LE.K)THEN
               L1 = K - J + 1
            ELSE
               L1 = J - K + 1
            END IF
            T(J,K) = TR1(L1)
   10    CONTINUE
   20 CONTINUE
      CALL EIGRS (T, N, EIG)
      RETURN
      END
```

CODE 7

```
      SUBROUTINE TOPEIG(TR1,N,NDIV2,APLUSB,EIG)
C
C     INPUT: TR1   = FIRST ROW OF TOEPLITZ MATRIX
C            N     = ORDER OF TOEPLITZ MATRIX
C            NDIV2 = N/2, THE ORDER OF SCRATCH MATRIX APLUSB
C            APLUSB= SCRATCH MATRIX OF ORDER NDIV2
C    OUTPUT: EIG   = ARRAY CONTAINING THE N EIGENVALUES
C
      DIMENSION APLUSB(NDIV2,NDIV2),EIG(N),TR1(N)
      SIGN = 1.0
      DO 40 I = 1,2
         IPOINT = (I - 1)*NDIV2 + 1
         DO 20 K = 1,NDIV2
            DO 10 J = 1,NDIV2
               L2 = N + 2 - K - J
               IF(J.LE.K)THEN
                  L1 = K - J + 1
               ELSE
                  L1 = J - K + 1
               END IF
               APLUSB(J,K) = TR1(L1) + TR1(L2)*SIGN
   10       CONTINUE
   20    CONTINUE
         CALL EIGRS (APLUSB, NDIV2, EIG(IPOINT))
         SIGN = -1.0
   40 CONTINUE
      RETURN
      END
```

Code 6 computes the eigenvalues of T directly, while Code 7 performs the mathematically equivalent computation of finding the eigenvalues of the smaller symmetric matrices A+B and A-B. Code 7 is computationally superior to Code 6, since it requires 75% less storage for matrices and executes 75% faster when matrix T is of sufficiently large order [3,4]. However, knowing only definition (13) of a real symmetric Toeplitz matrix T and that Code 7 computes the eigenvalues EIG of matrix T (if it is of even order n) by using a canned eigenvalue routine EIGRS for any real symmetric matrix, a maintenance programmer may be puzzled by Code 7 and tempted to replace it with the shorter and more comprehensible (but less efficient) Code 6.

As a final example, consider approximating the scaled Airy function exp(zta) Ai(z) for values of z of large magnitude by evaluating a partial sum of the asymptotic series [5]

$$\exp(\mathrm{zta})\ \mathrm{Ai}(z) \sim (\pi^{-0.5}\, z^{-0.25}/2) \sum_{k=0}^{\infty} (-1)^k c_k\, \mathrm{zta}^{-k} \qquad (14)$$

where

$$\mathrm{zta} = z^{1.5}/1.5, \quad \mathrm{abs}(\arg z) < \pi$$

$$c_0 = 1, \quad c_{k+1}/c_k = k/2 + 5(k+1)^{-1}/72 \qquad (15)$$

Writing

$$G_k = (-1)^k c_k\, \mathrm{zta}^{-k}$$

we see that the terms of the sum in (14) can be generated recursively:

$$G_{k+1} = -(c_{k+1}/c_k)\, \mathrm{zta}^{-1}\, G_k \quad (k = 0,1,2, \ldots) \qquad (16)$$

and, a fortiori,

$$\mathrm{abs}(G_{k+1}) = \mathrm{abs}(\mathrm{zta}^{-k-1}) \prod_{j=0}^{k} (c_{j+1}/c_j) \doteq 2^{-k}\, k!\, \mathrm{abs}(\mathrm{zta}^{-k-1})5/72 \qquad (17)$$

For any specific value of z, the approximation to the scaled Airy function having maximum accuracy is obtained by evaluating the partial sum

$$\sum_{k=0}^{n} G_k \qquad (18)$$

where $G_{n+1}$ is the first term encountered in the series for which

$$\mathrm{abs}(G_{n+1}) > \mathrm{abs}(G_n) \qquad (19)$$

By (16), condition (19) is equivalent to

$$c_{n+1}/c_n > m = \text{abs}(zta) \tag{20}$$

Therefore, the first value of n for which (19) holds is

$$n = [(9m^2 + 9m + 1)^{0.5}/3 + m - 0.5] \sim [2m - (14.4m)^{-1}] \tag{21}$$

where [x] is the smallest integer greater than or equal to x. However, if m is sufficiently large computation of $G_k$ will cause computer underflow well before k = n, and the evaluation of (18) out to $G_n$ then becomes impractical. Consequently, in a practical computation, (18) should be evaluated out to $G_n$ or the $G_k$ with the smallest magnitude that does not cause computer underflow, whichever comes first.

The largest value of m for which (18) can be summed out to $G_n$ without any of the $G_k$ underflowing depends on the smallest positive computer number s. A good approximation to this value of m is obtained by setting the approximation (17) for abs($G_{k+1}$) equal to s with k = 2m, applying Stirling's asymptotic formula for factorials and then solving the resulting equation for m. This gives the equation

$$m = -\ln(m)/4 - \ln(7.2s\pi^{-0.5})/2 \tag{22}$$

whose solution is m*, the limit of the rapidly convergent sequence

$$m_j = -\ln(m_{j-1})/4 + m_0, \quad m_0 = -\ln(7.2s\pi^{-0.5})/2 \tag{23}$$

In other words, for any given value of z for which m does not exceed m*, (18) can be summed out to $G_n$ without any of the $G_k$ underflowing; if m exceeds m*, (18) can be summed out to the term whose index k is $[2m^*f - (14.4m)^{-1}]$ just short of $G_k$ underflowing for a positive value of f < 1.

To find f we proceed in the same way that we did to find m*, but we set the approximation (17) equal to s with k = 2m*f. This gives

$$f = (\,(f + 1/4m^*)\ln(f) - c\,) \,/\, \ln(em/m^*) \qquad (24)$$

$$c = (1/2m^*)\ln(\,7.2sm \,/\, \sqrt{\pi m^*}\,)$$

whose solution is the limit of the sequence

$$f_j = (\,(f_{j-1} + 1/4m^*)\ln(f_{j-1}) - c\,) \,/\, \ln(em/m^*) \qquad (25)$$

for an initial value $f_0 < 1$. If $f_0 = 1/\log(m)$, then $f_2$ gives an adequate approximation to f.

To summarize, evaluating the asymptotic series until (19) is satisfied or $G_k$ underflows is similar to evaluating the series out to the term whose index k has the value $[2m^*f-(14.4m)^{-1}]$, where $f = m/m^*$ if $m \leq m^*$. For any given z, the latter method provides an apriori estimate of the optimal number of terms of the series to sum, eliminating the need for the comparison test (19) during the summation of the series.

Therefore, two possible structured FORTRAN subprograms for evaluating the asymptotic series are

CODE 8

```
      FUNCTION SUM (ZTA)
      COMPLEX*16 SUM, ZTA, GK, RZTA
      DOUBLE PRECISION ABSGK, ABSGK1, S, FACTOR
      DATA S/2.94D-39/
      SUM = DCMPLX(0.0D0, 0.0D0)
      K = 0
      GK = DCMPLX(1.0D0,0.0D0)
      RZTA = GK / ZTA
      ABSGK1 = 1.0D0
C     DO UNTIL ( ABSGK1.GE.ABSGK .OR. ABSGK1.LE.S)
    5    SUM = SUM + GK
         FACTOR = 0.5D0 * K  +  5.0D0 / (K + 1) / 72.0D0
         K = K + 1
         GK = - FACTOR * RZTA * GK
         ABSGK = ABSGK1
         ABSGK1 = CDABS(GK)
         IF ( ABSGK1.LT.ABSGK .AND. ABSGK1.GT.S)GO TO 5
      CONTINUE
      RETURN
      END
```

CODE 9

```
      FUNCTION SUM (ZTA)
      COMPLEX*16 SUM, ZTA, GK, RZTA
      DOUBLE PRECISION FACTOR
      REAL MSTAR, EOMSTR, TMSTAR, R4MSTR, CLN, M, F, C, D
      DATA MSTAR/42.72/, TMSTAR/85.44/, EOMSTR/.06363019/
      DATA CLN/-89.198027/, R4MSTR/5.8520599E-03/
      M = CDABS(ZTA)
      IF(M .GT. MSTAR)THEN
         F = 1.0/ALOG10(M)
         C = (CLN + ALOG(M))/TMSTAR
         D = ALOG(EOMSTR*M)
         F = ((F + R4MSTR)*ALOG(F) - C)/D
         F = ((F + R4MSTR)*ALOG(F) - C)/D
         N = 2.0 * (MSTAR * F + 0.5) - .069444444 / M
      ELSE
         N = 2.0 * (M + 0.5) - .069444444 / M
      END IF
      SUM = DCMPLX(1.0D0, 0.0D0)
      GK = SUM
      RZTA = GK / ZTA
      DO 100 K = 0, N-1
         FACTOR = 0.5D0 * K + 5.0D0 / (K + 1) / 72.0D0
         GK = - FACTOR * RZTA * GK
         SUM = SUM + GK
  100 CONTINUE
      RETURN
      END
```

Code 8 terminates summation of the series when either (19) is satisfied or a term of the series underflows s, while Code 9 terminates when an apriori estimate of the summation index is reached for which either one of these conditions is satisfied. Although Code 9 is longer than Code 8, the VAX 11/780 executes Code 9 30% faster than Code 8; in fact, if an initial value for the sequence (25) were found so that $f_1$ would give an adequate approximation to f, then Code 9 would execute approximately 40% faster than Code 8. In any event, knowing only that the purpose of these codes is to evaluate the sum (18) for a given value of zta until either (19) is satisfied or $G_k$ underflows s, without additional information a maintenance programmer would have difficulty comprehending how this is accomplished by the more efficient Code 9.

## CONCLUSION

Although structured coding promotes software clarity by highlighting the flow of control of the code, the result of searching for a computational method that hopefully performs the same function optimally may demote software comprehensibility. Since the algorithm that is finally implemented can depart significantly from the formula that was used originally to define the purpose of the code, the algorithm must be documented sufficiently or else the code may be difficult to maintain effectively.

## REFERENCES

1. Marvin Goldstein and John Lawson, Jr., "An Example of Quality Mathematical Software," NUSC TM No. 811044, 15 April 1981

2. Marvin Goldstein, "Efficient Matrix Addressing On Virtual Memory Machines," NUSC TM No. 801044, 1 April 1980

3. Marvin Goldstein and William Babson, "A Subroutine Package for the Efficient Solution of the Eigenproblem of Real Symmetric Toeplitz Matrices," NUSC TM 831172, 30 December 1983

4. Marvin Goldstein, "Reduction of the Eigenproblem for Hermitian Persymmetric Matrices," Math. Comp., v. 28, no. 125, 1974 Jan.

5. M. Abramowitz and I. Stegun, Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables, NBS Applied Math. Series 55, 1964

MATHEMATICALLY EQUIVALENT, COMPUTATIONALLY NON-EQUIVALENT FORMULAS AND SOFTWARE COMPREHENSIBILITY
TM 851062
Marvin J. Goldstein
Surface Ship Sonar Department
22 May 1985
UNCLASSIFIED

DISTRIBUTION LIST

| | | | |
|---|---|---|---|
| Code 00 | CAPT Ailes | 3233 | R. Christman |
| 01 | E.L. Messere | 33 | L. Freeman |
| 01A | R. Moore | 3301 | T. Bateman |
| 01Y | Dr. J. Short | 33A | B. Cole |
| 10 | Dr. W. VonWinkle | 33A3 | Dr. W. Roderick |
| 101 | Dr. E. Eby | 33B3 | D. Counsellor |
| 101 | F. Weigle | 3311 | J. Higgs |
| | Dr. L. Goodman | 3312 | C. Becker |
| | Dr. K. Lima | 3312 | P. Saikowski |
| 02 | Dr. C. Kindilien | 3313 | J. Gregor |
| 0211 | R. Bernier | 3313 | D. Aker |
| 021311 | NL Library (3 copies) | 3313 | P. Anchors |
| 021312 | NPT Library | 3313 | N. Suliniski |
| 03 | D. Walters | 3313 | D. Fingerman |
| 0303 | G. Hill | 3313 | R. Molino |
| 20 | W. Clearwaters | 3314 | Dr. C. Carter |
| 32 | Dr. J. Kingsbury | 3314 | Dr. R. Dwyer |
| 3211 | J. O'Sullivan | 3314 | Dr. A. Nuttall |
| 3211 | W. Babson | 3314 | W. Goldman |
| 3211 | A. Lesick | 3314 | I. Cohen |
| 3211 | Dr. N. Owsley | 3314 | R. Johnson |
| 3212 | S. Dzerovych | 3314 | J. Sikorski |
| 3212 | Dr. J. Ianniello | 3314 | P. Stahl |
| 3212 | J. Pearson | 3314 | R. Trembley |
| 3212 | J. Ferrie | 332 | Dr. R. Radlinski |
| 3213 | S. Kessler | 3321 | F. P. Fesseden |
| 3213 | W. Axtell | 333 | W. Schumacher |
| 3213 | R. Cox | 3331 | D. Browning |
| 3213 | D. Rawson | 3331 | P. Fisch |
| 3213 | L. C. Ng | 3331 | J. Chester |
| 3213 | J. Sanchis | 3331 | W. Hauck |
| 3232 | G. Connolly | 3331 | P. Koenigs |
| 3232 | Dr. R. Streit | 3331 | M. Fecher |
| 3232 | B. Helme, Jr. | 3331 | J. Monti |
| 3233 | Dr. H. Schloemer | 3331 | J. Syck |
| 3233 | Dr. S. Ko | 3332 | Dr. H. Weinberg |
| 3233 | Dr. W. Strawderman | 3332 | G. Botseas |
| 3234 | D. T. Porter | 3332 | Dr. D. Lee |
| 325 | W. Coggins | 3332 | M. Goldstein (15) |
| 325 | J. Shores | 3332 | W. Kanabis |
| 3251 | D. Daros | 3332 | H. Sternberg |
| 3251 | C. Bowman | 3332 | R. Deavenport |
| 3251 | P. Miner | 3332 | Dr. D. Wood |
| 3252 | T. Anderson | 3332 | R. Drinkard |
| 3253 | J. Ionata | 3332 | J. Nordquist |
| 3253 | M. Kuznitz | 3332 | E. Robinson |
| 3253 | J. Munoz | 3332 | E. Jensen |
| 3253 | R. Leask | 3332 | C. Turner |
| 3202 | D. Yarger | 3332 | L. Petitpas |

TM 851062

| | |
|---|---|
| 3333 | C. Brown |
| 3333 | G. Brown |
| 3333 | F. Farmer |
| 3333 | E. Montavon |
| 3333 | J. Prentice |
| 3333 | W. Sternberg |
| 3333 | W. Wachter |
| 3333 | L. Walker |
| 3334 | J. Hall |
| 3334 | E. Gannon |
| 3334 | V. Edwards |
| 3334 | D. Klingbeil |
| 3334 | F. McMullen |
| 34 | Dr. D. Dence |
| | J. Katan |
| | Dr. D. Fessenden |
| | K. Hafner |
| | J. Casey |
| | A. Bruno |
| | D. Dixon |
| 35 | D. Cardin |
| | T. Conrad |
| | L. Cabral |
| 36 | Dr. J. Sirmalis |
| | J. Griffin |
| | D. Blundell |
| | M. Lydon |
| | S. Wax |
| | S. Ashton |
| | R. McMahon |
| | S. Meyers |
| | K. Padolino |
| 37 | C. Curtis |
| | Q. Huynh |
| 38 | J. Kyle |
| 401 | A. Carlson |
| | J. Clark |
| | Dr. A. Kalinowski |
| | Dr. R. Kasper |
| | R. Manstan |
| | R. Munn |
| | C. Nebelung |
| | Dr. J. Patel |
| | B. Radley |
| | A. Shigematsu |
| | Dr. M. Tucchio |
| 402 | M. Berger |
| 4111 | S. Horvitz |
| 4331 | B. Antrim |
| | S. Walsh |
| 434 | K. Steele |
| | G. Lussier |
| 60 | Dr. J. Cohen |
| 70 | G. Bain |
| 701 | G. Elias |
| 72 | G. Daglieri |
| | R. Wilson |
| | S. Schneller |
| | J. Auwood |
| | T. Wheeler |
| | P. Breslin |
| | A. Alfiero |
| | J. Gribbin |
| 73 | S. Capizzano |
| | R. Pingree |
| | D. Quigley |
| 74 | M. Lee |
| | C. Brockway |
| | B. Sullivan |
| | R. Hoy |
| | A. Blau |
| | T. Perella |
| | W. Cote |